Robotics Nanodegree

Deep Learning Project

In this project, we'll use the deep neural network to identify and track a target in simulation. So-called "follow me" applications like this are key to many fields of robotics and the very same techniques you apply here could be extended to scenarios like advanced cruise control in autonomous vehicles or human-robot collaboration in industry.

If you need further information, assistance or referral about a project issue, please contact kiang.ng@hotmail.com.

UDACITY

UDACITY

Ng Fang Kiang
Kiang.ng@hotmail.com
Instagram.com/jorcus96
linkedin.com/in/jorcus
fb.com/ngfangkiang
github.com/jorcus

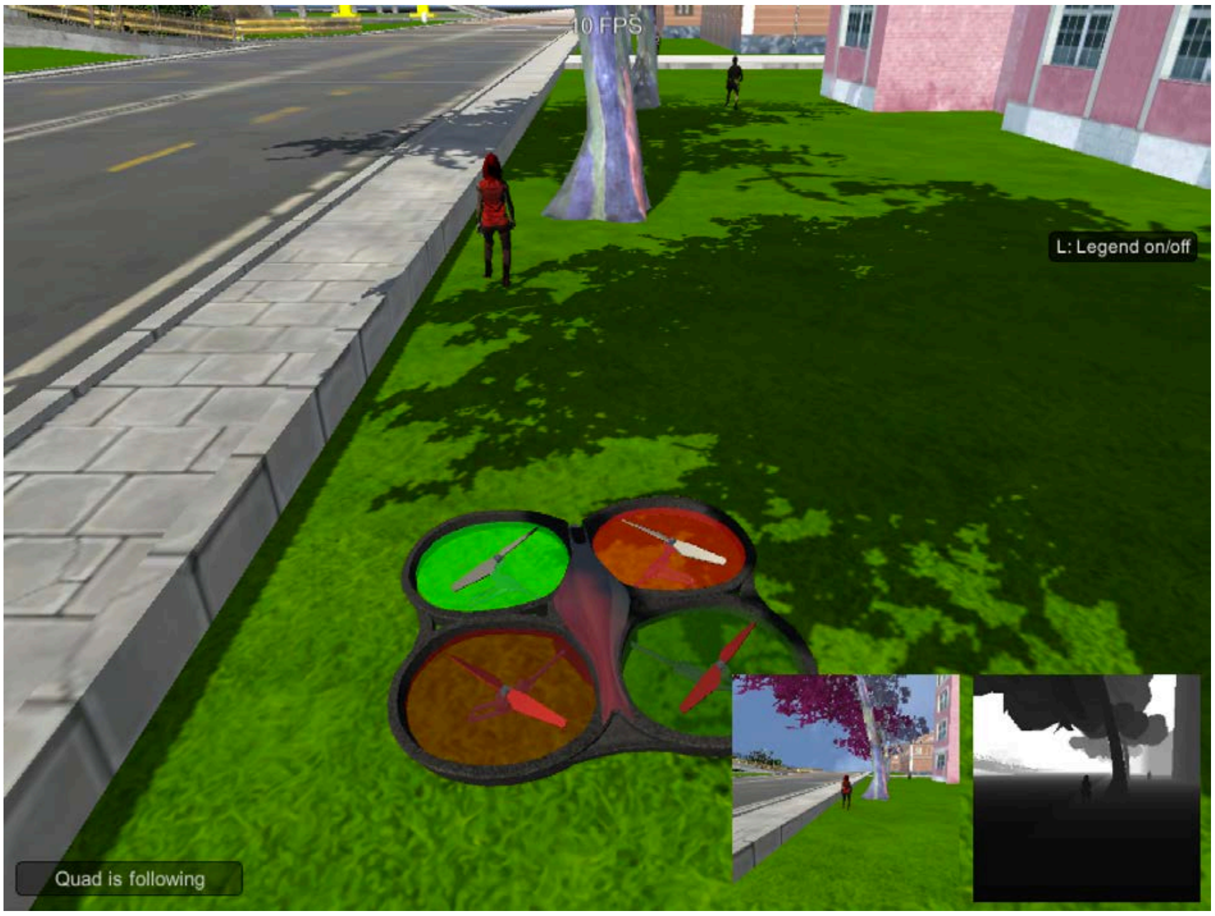
Ng Fang Kiang

Project 4: Follow Me

Prepared/Updated : 19 Nov 2017

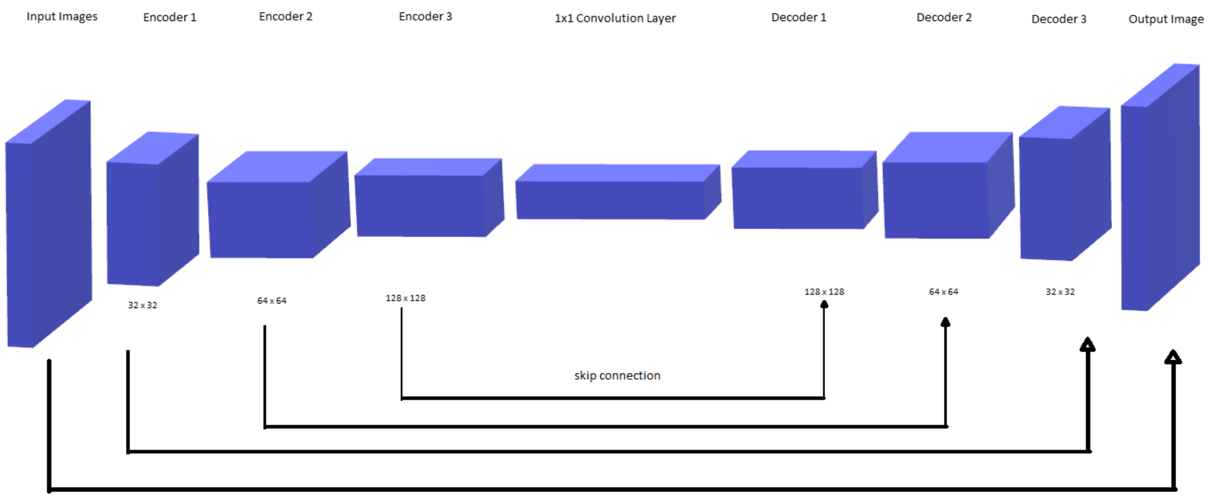
Deep Learning Project

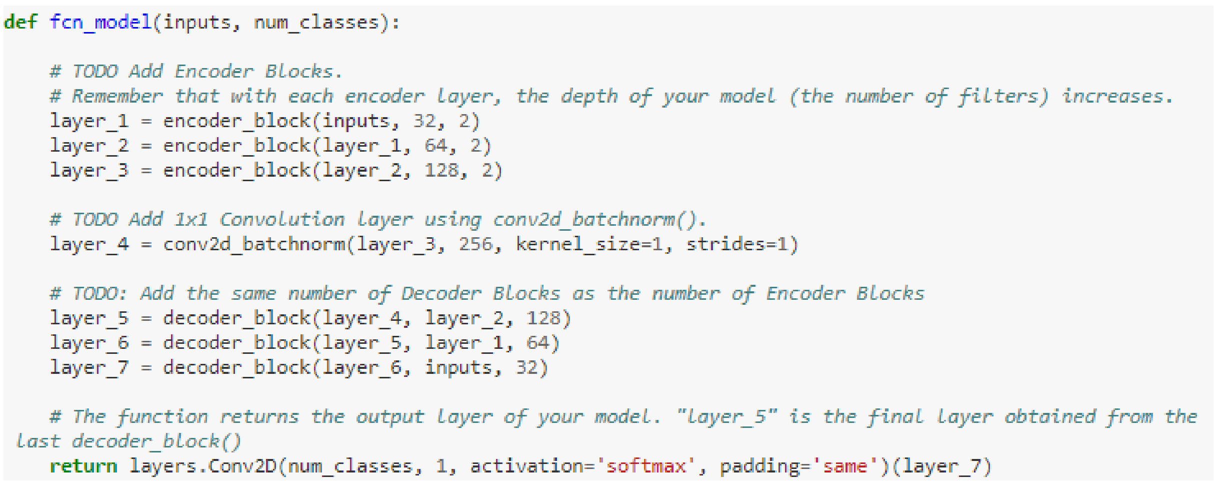
In this project, we will train a deep neural network to identify and track a target in simulation. So-called "follow me" applications like this are key to many fields of robotics and the very same techniques you apply here could be extended to scenarios like advanced cruise control in autonomous vehicles or human-robot collaboration in industry. Here's the video result.



Network Architecture

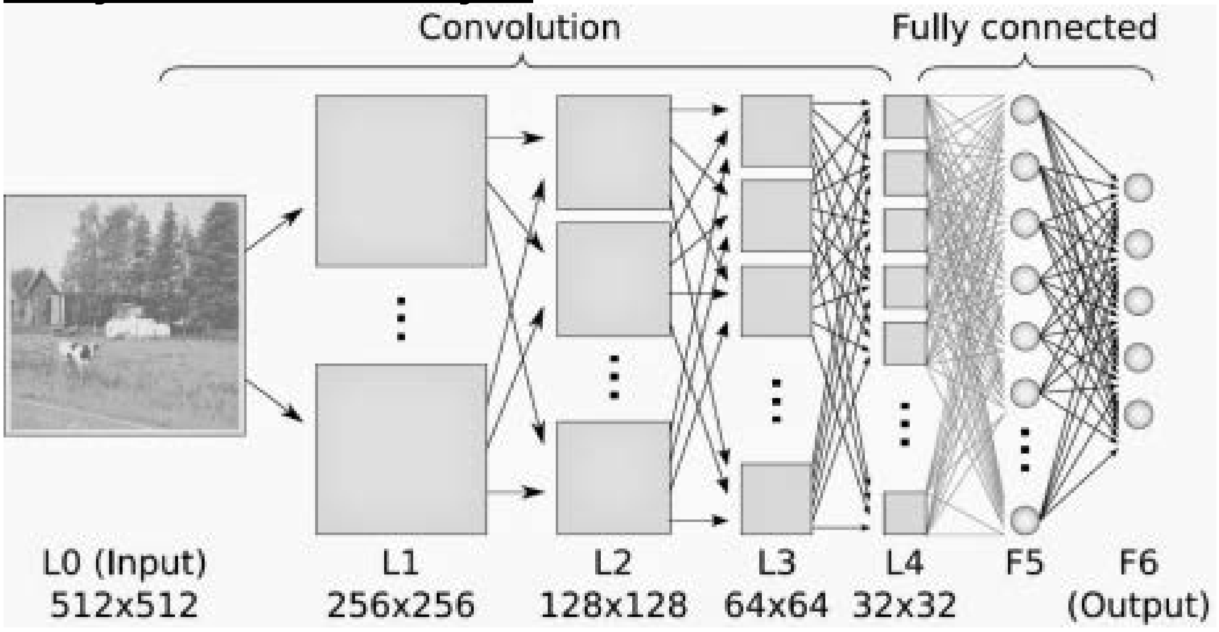
I initiate with the 2 encoder layers, 1x1 convolution and then 2 decoder layers and it fails to detect the target when its far away from the quadrotor. Then, I increase each encoder and decorder layers, and finalize the model architecture as 3 encoder layers, the 1x1 convolution with 256 filters, and then 3 decoder layers as following...





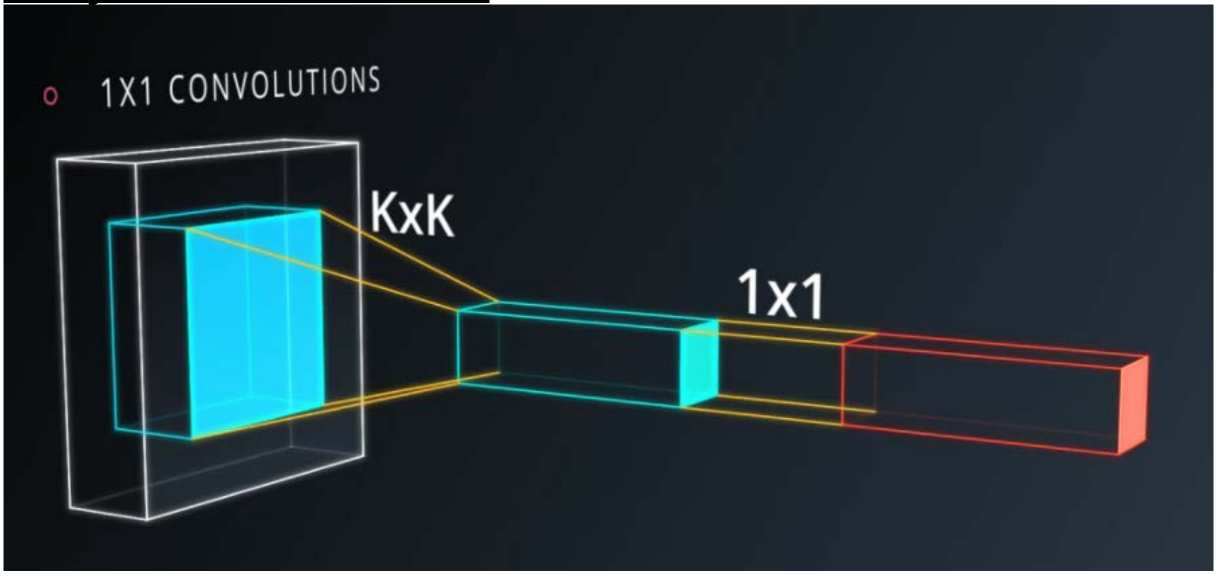
```
def fcn_model(inputs, num_classes):

    # TODO Add Encoder Blocks.
    # Remember that with each encoder layer, the depth of your model (the number of filters) increases.
    layer_1 = encoder_block(inputs, 32, 2)
    layer_2 = encoder_block(layer_1, 64, 2)
    layer_3 = encoder_block(layer_2, 128, 2)

    # TODO Add 1x1 Convolution layer using conv2d_batchnorm().
    layer_4 = conv2d_batchnorm(layer_3, 256, kernel_size=1, strides=1)

    # TODO: Add the same number of Decoder Blocks as the number of Encoder Blocks
    layer_5 = decoder_block(layer_4, layer_2, 128)
    layer_6 = decoder_block(layer_5, layer_1, 64)
    layer_7 = decoder_block(layer_6, inputs, 32)

    # The function returns the output layer of your model. "layer_5" is the final layer obtained from the
 last decoder_block()
    return layers.Conv2D(num_classes, 1, activation='softmax', padding='same')(layer_7)
```

Fully Connected Layer



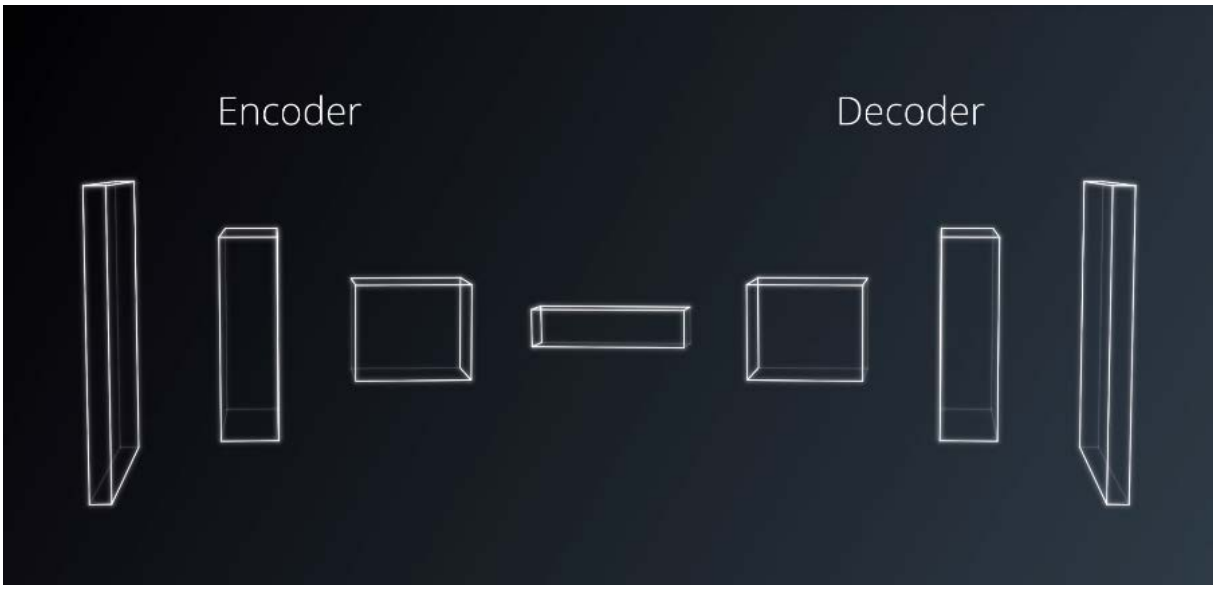
In principle, fully connected layer are same as the traditional multi-layer perceptron neural network (MLP). The fully connected layer are generally consists at least three or more layers which is an input, an output and at least one hidden layers. It connected every single neuron from the previous layer to the next layer and this is the reason people name it as “fully connected layer”. In fully connected layers, it doesn’t preserve spatial information. In fact, we can integrate convolutional directly into the layer to create fully convolutional network(FCN) with the change from fully connected layer to convolutional layer because convolution layer preserve spatial information throughout the entire network.

1-by-1 Convolutions



In our fully convolutional networks, we noticed that fully connected layer doesn't preserve any spatial informations. So, we implement 1x1 convolutional layer to preserve the spatial informations of the input. With implementation this technique, it also can be use to change the filter space dimensionality of the layer either increase or reducing the dimensionality of the layer. So that, A fully-connected layer of the same size would result in the same number of features. However, replacement of fully-connected layers with convolutional layers presents an added advantage that during inference (testing you mode), you can feed images of any size into your trained network.

Encoder and Decoder



The encoder is usually a pre-trained classification network like VGG/ResNet followed by a decoder network. The decoder network/mechanism is mostly where these architecture differs. The task of the decoder is to semantically project the discriminative features (lower resolution) learned by the encoder on the pixel space (higher resolution) to get a dense classification. Sources

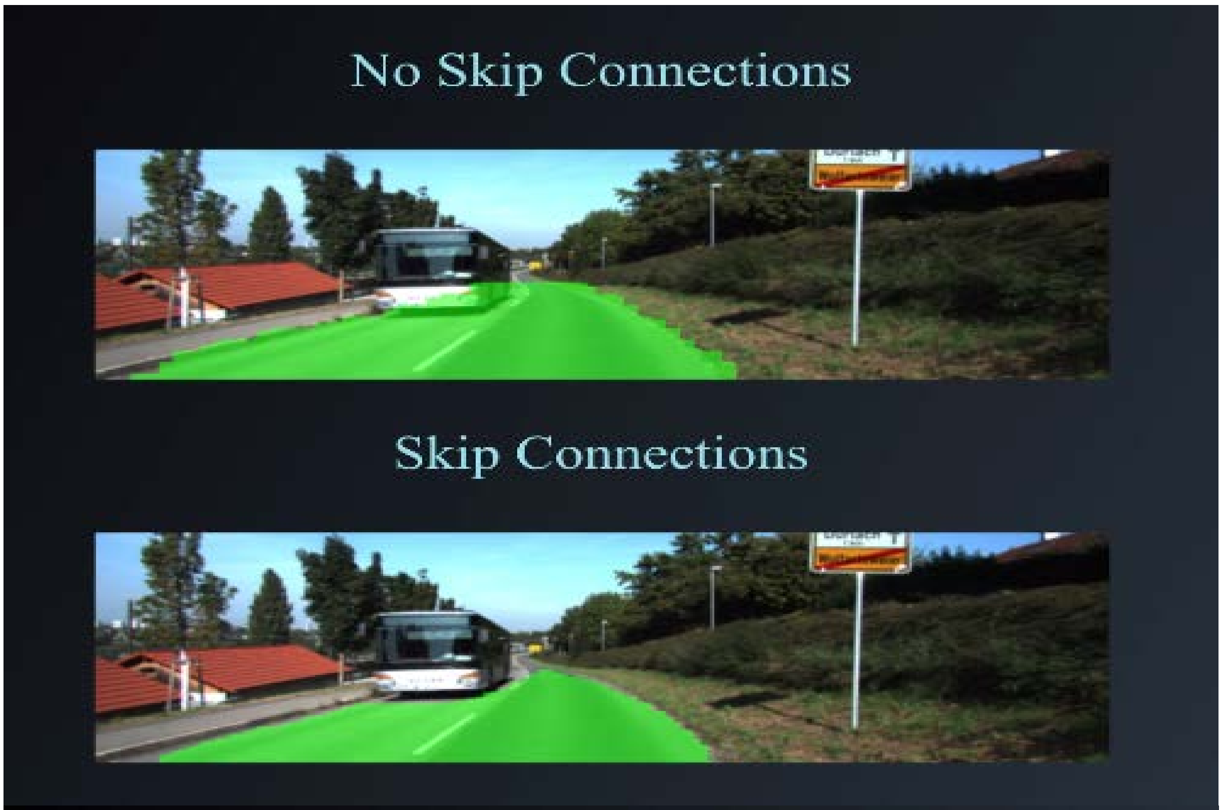
The goal of encoder and decoder

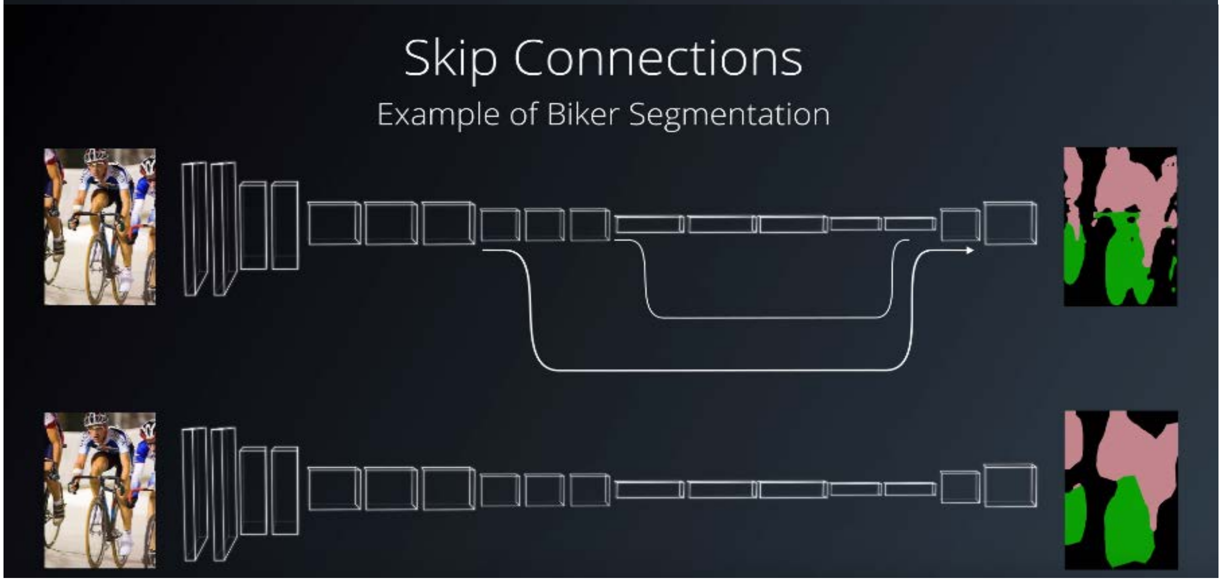
Encoder : To extract features from the image.

Decoder : To up-scales the output of the encoder to the same size as the original image the features

Skip Connections

Skip connections allow the neural networks to use information from multiple resolutions scales or resolutions. In the above we encoder the image to extract the features from the images and decoder back the output of the encoder back to the original size, but some of informations might be lost. We can implementation of skip connections to retain the information easily. The bellows shows the difference between with and without implementation of skip connections.





Hyperparameters

I've tuned a few scenario of hyperparameters and this is my final decision because it is able to detect an object that is far away from the vision.

Learning rate : 0.002 (From my observation when tuning the learning rate, It resulted the neural networks learned faster when higher learning rate is implemented. Due to I'm going to use high numbers of learning rate, so I a minor change increased from 0.001 to 0.002.)

Batch size: 32 (Stay Default)

Numbers of Epoch : 100 (This numbers of approached is abit high, it might cause overfitting. However, I still decided to select this because I'm going to modified this project when I have time)

Steps per Epoch : 200 (Stay Default)

Validation Steps : 50 (Stay Default)

Workers : 5

Training Machines

Deep learning is a high computation neural networks which don't run efficiently in CPU. So, I decided to use my GPU (Nvidia GTX 1070) which faster than my CPU more than 10 times to train this fully convolution neural-networks. The below shows my PC specs that I used for training.

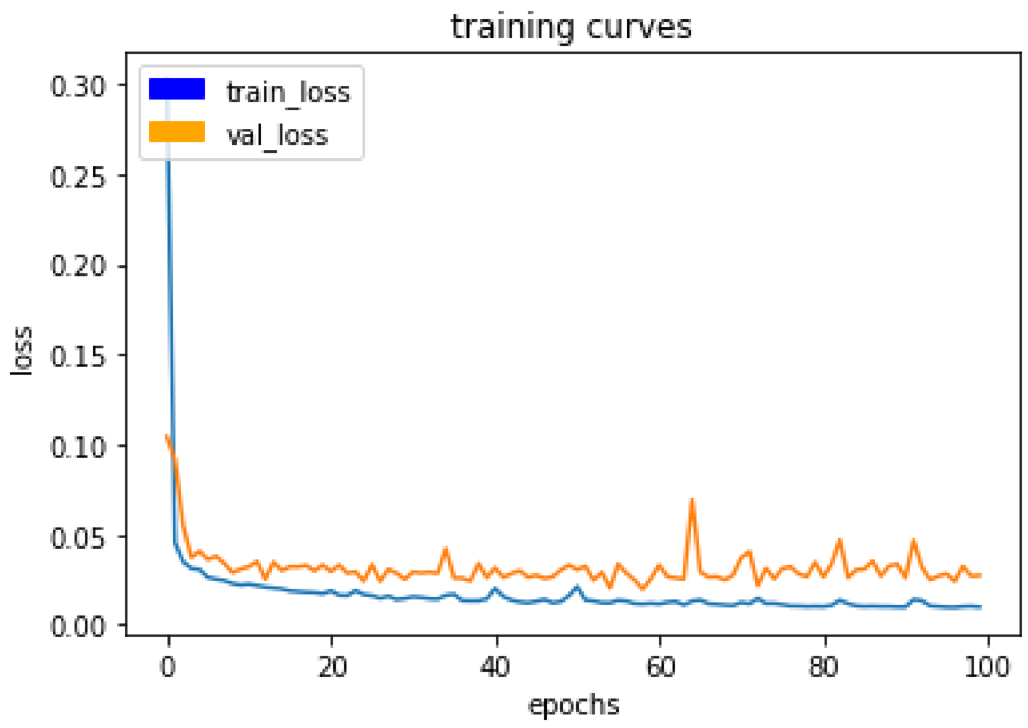
CPU : Intel I7 6700K

GPU : Nvidia GTX 1070

RAM : 32GB

Average times per epoch : 55 sec

Result



```
# Scores for while the quad is following behind the target.
true_pos1, false_pos1, false_neg1, iou1 = scoring_utils.score_run_iou(val_following, pred_following)
```

```
number of validation samples intersection over the union evaulated on 542
average intersection over union for background is 0.995990646239
average intersection over union for other people is 0.36794329476
average intersection over union for the hero is 0.916386351051
number true positives: 539, number false positives: 0, number false negatives: 0
```

```
# Scores for images while the quad is on patrol and the target is not visable
true_pos2, false_pos2, false_neg2, iou2 = scoring_utils.score_run_iou(val_no_targ, pred_no_targ)
```

```
number of validation samples intersection over the union evaulated on 270
average intersection over union for background is 0.986185149222
average intersection over union for other people is 0.718848343485
average intersection over union for the hero is 0.0
number true positives: 0, number false positives: 63, number false negatives: 0
```

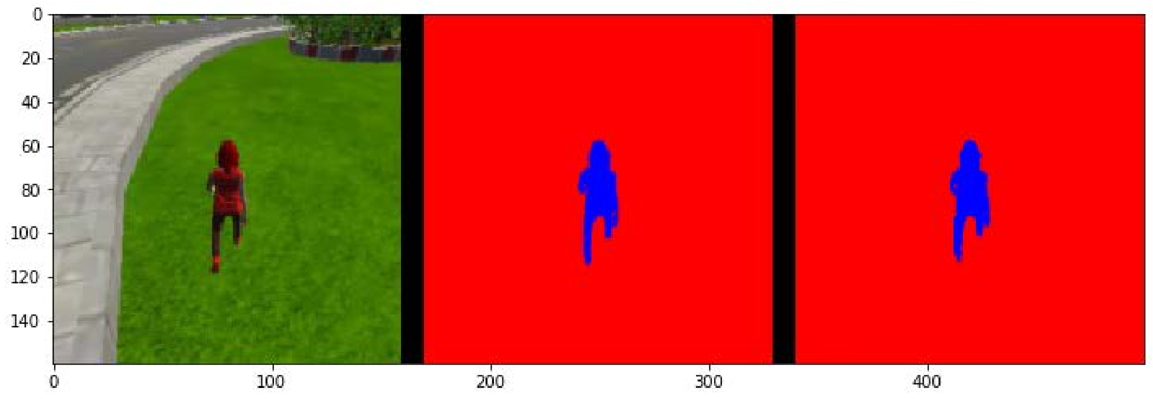
```
# This score measures how well the neural network can detect the target from far away
true_pos3, false_pos3, false_neg3, iou3 = scoring_utils.score_run_iou(val_with_targ, pred_with_targ)
```

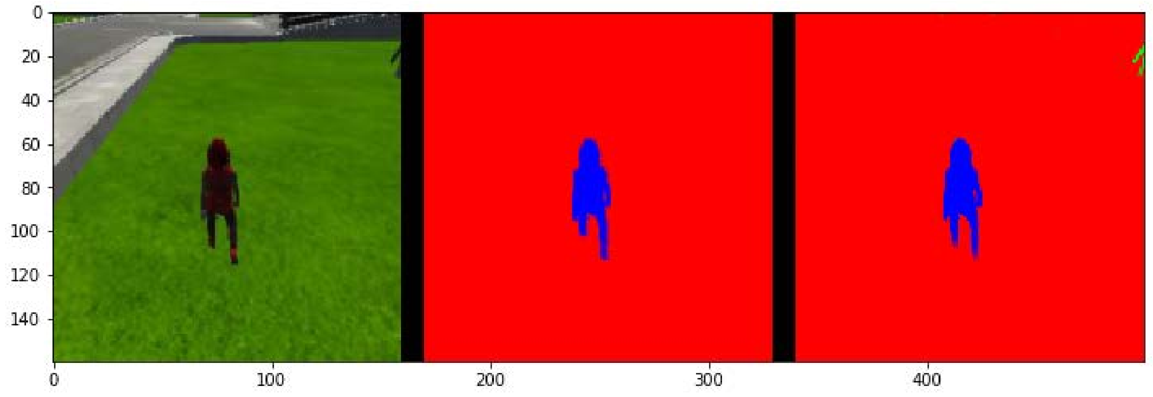
```
number of validation samples intersection over the union evaulated on 322
average intersection over union for background is 0.996499004335
average intersection over union for other people is 0.453317615995
average intersection over union for the hero is 0.215140560149
number true positives: 124, number false positives: 2, number false negatives: 177
```

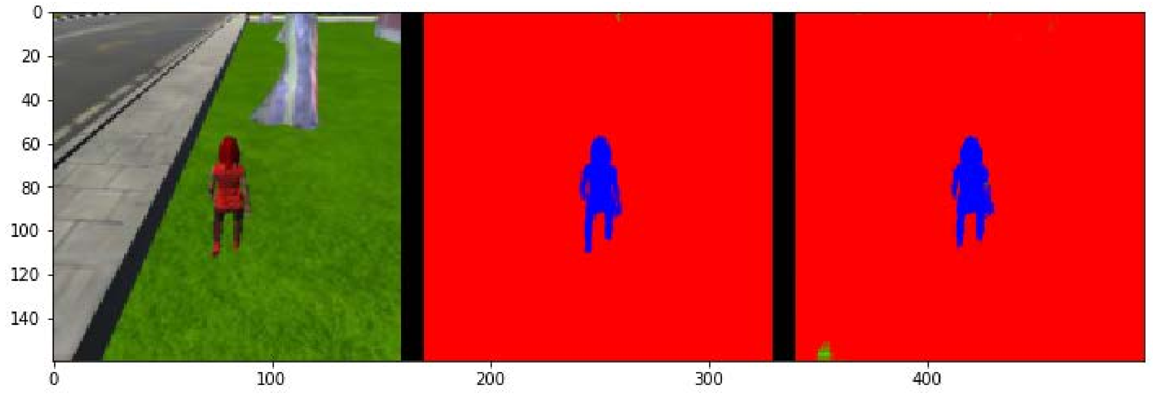
Final Score : 0.41

Images while following the target

This works pretty good while there's a tiny errors exists on the third image.

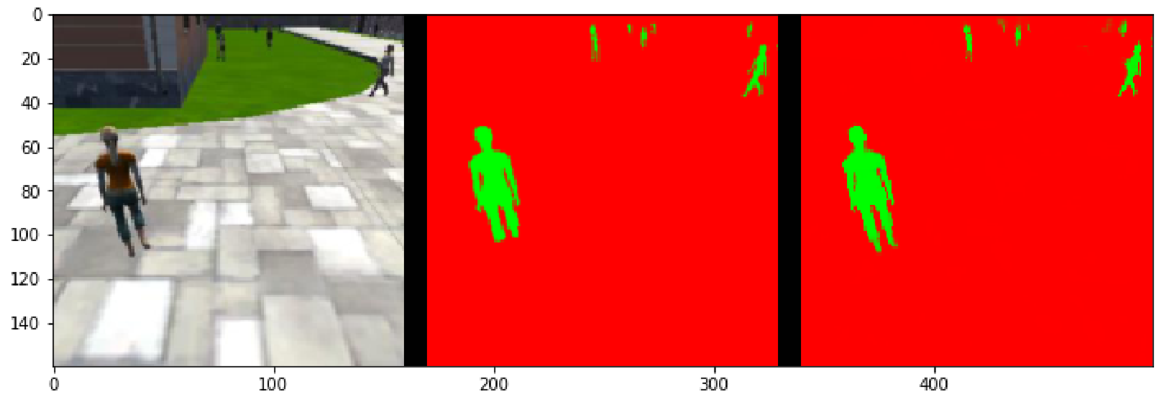


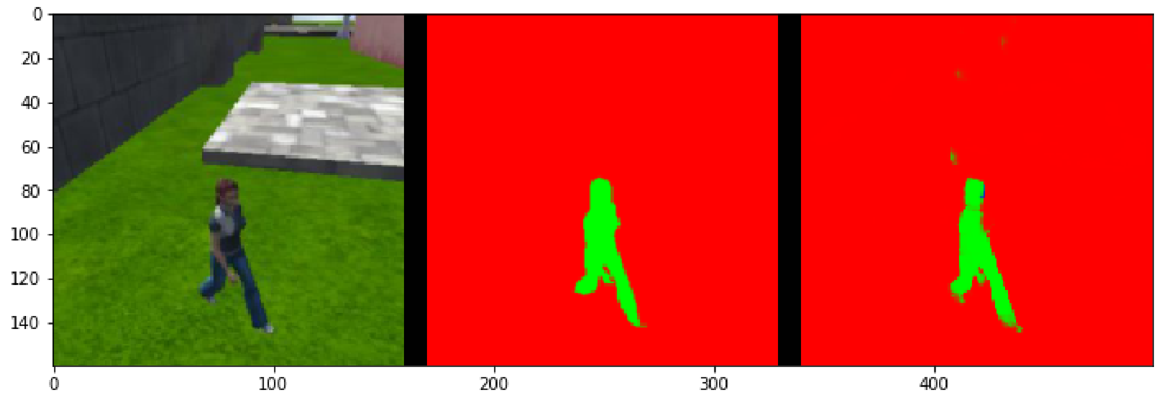


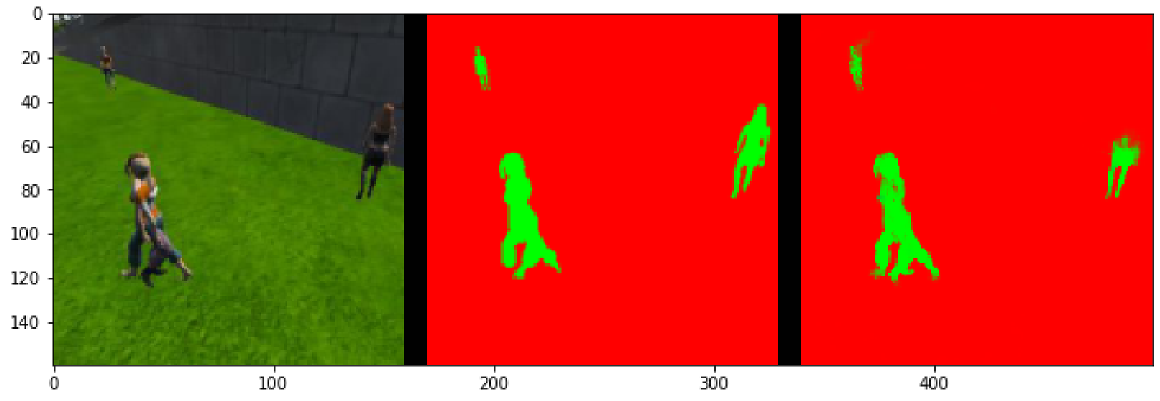


Images while at Patrol without target

As you can see from the image below, this works pretty well and detect nothing when the patrol without target!

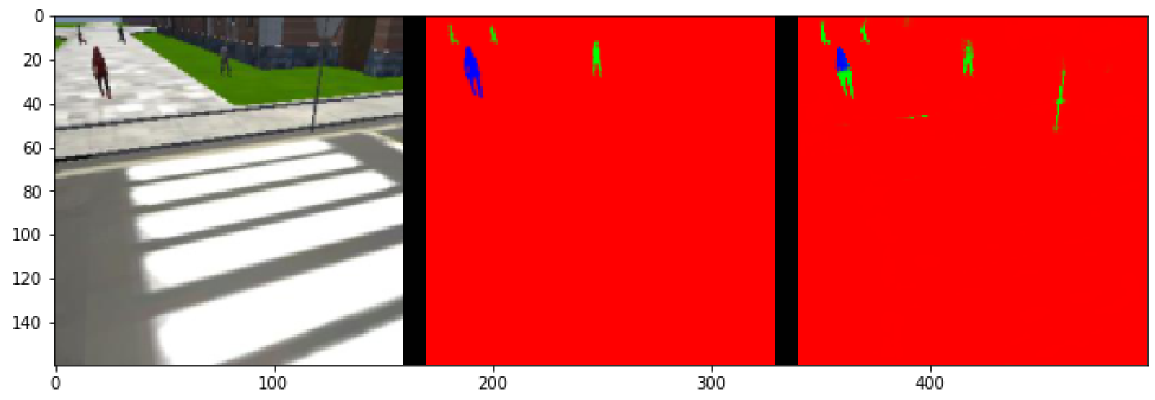
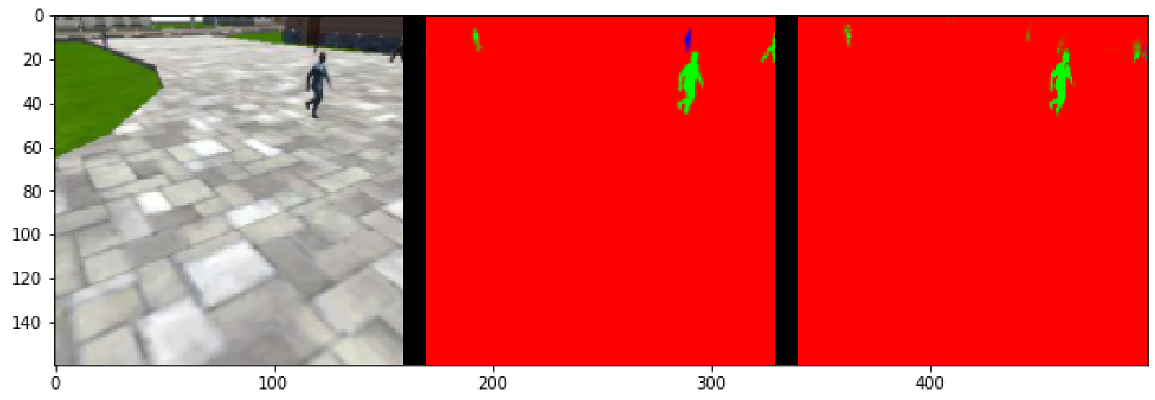
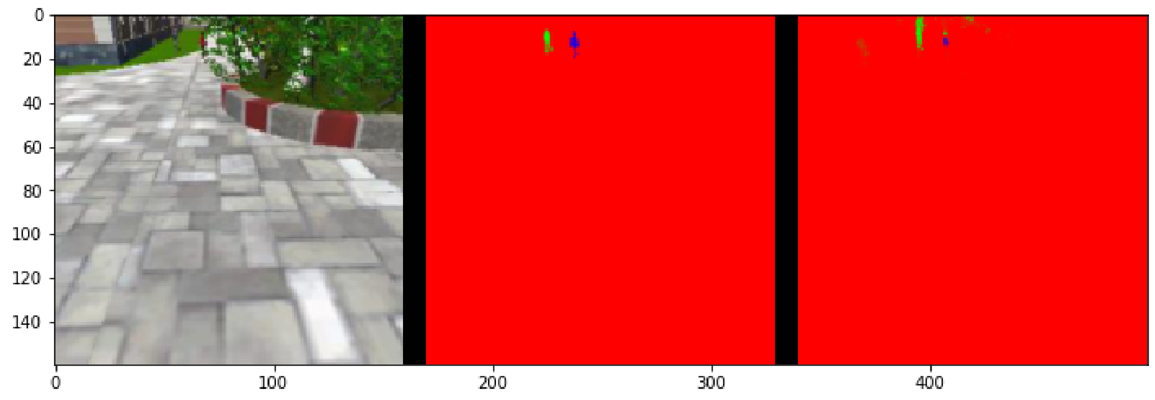






Images while at patrol with target

This is challenging when detecting the target at very far away from the quadrotor vision. I feel surprise it works to detect the target.

Future Enhancement

Nothing is perfect, there's always lots of fun works to improve.

1. Model Architecture and the Hyperparameters can be improve because the result is just slightly pass requirements

2. Implementation Data augmentations technique to increase the numbers of data

3. Utilize more Visualize tools for further analysis

4. Collecting others target data likes dogs, cats or other animals to train difference segmentation. I think our current model might work when the object is close enough since the object is looks small which the model may find difficulty when segmentation. From the other hand, the big object may works well as I think. Maybe we can implement the bounding boxes recognition in this project too?

5. The training curve appears that the model is overfitting with final epoch of training loss is 0.0098 and validation loss is 0.0274. Increasing the number of data might improve it.